RMI Based Multi-Area Power System Load Flow Monitoring

K. Nithiyananthan and V. Ramachandran

Abstract—**The main objective of this paper is to construct a distributed environment through which the load flow solutions of multi-area power systems can be monitored and controlled. A single-server/multi-client architecture which enables the neighboring power systems to access the remote load flow server at any time, with their respective data and to get the load flow solutions from the remote server has been proposed. An RMI (Remote Method Invocation) based distributed environment has been implemented in such a way that for every specific period of time, the remote server obtains the system data simultaneously from the neighboring power systems which are the clients registered with the remote load flow server and the load flow solutions from the server have been sent back to the respective clients. The load flow server creates a new thread of control for every client's request and hence complete distributed environment is exploited.**

Index Terms—**Distributed computing, power systems, load flow, RMI, client-server model.**

I. Introduction

The power system load flow solution obtained through conventional client-server architecture is complicated, memory management is difficult, source code is bulky, and exception-handling mechanism is not so easy. In the conventional power system operation and control, it is assumed that the information required for monitoring and controlling of power systems is centrally available and all computations are to be done sequentially at a single location [1]. With respect to sequential computation, the server has to be loaded every time for each client's request and the time taken to deliver the load flow solution is also comparatively high [2]-[4].

This paper outlines a new approach to develop a solution for load flow analysis by way of distributed computing. RMI based client-server architecture overcomes the difficulties associated with sequential computation and it can be easily implemented.

RMI uses built-in java security mechanism and hence the distributed load flow monitoring through an applet definitely secures the safety of the server as well as power system data transfer.

The structure of this paper is as follows. Section II presents the RMI based architecture for automated load flow solutions. In Section III an algorithm is described for load flow monitoring based on RMI. The results of software implementation are presented in Section IV. Finally, Section V concludes the paper.

Manuscript received December 31, 2002; revised November 25, 2003.

K. Nithiyananthan is with The Department of Electrical and Electronics Engineering, College of Engineering, Guindy, Anna University, Chennai-25, India (e-mail: knithiyananthan@hotmail.com).

Dr.V.Ramachandran is with The Department of Computer Science and Engineering, College of Engineering, Guindy, Anna University, Chennai-25, India (e-mail: rama@annauniv.edu).

Publisher Item Identifier S 1682-0053(04)0194

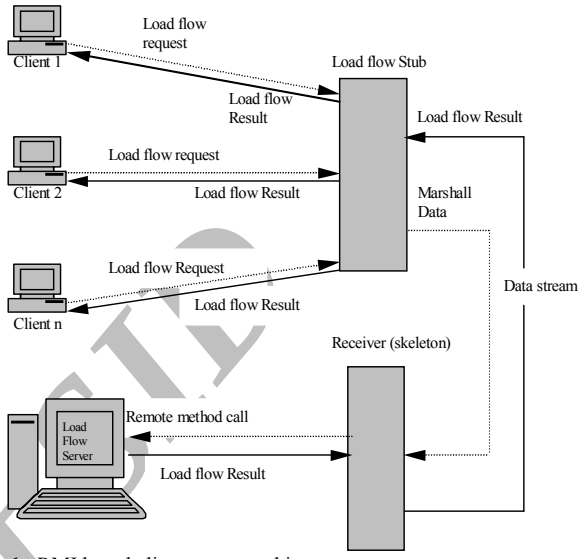

Fig. 1. RMI based client-server architecture.

II. RMI Based Architecture for Automated Load Flow Solutions

In the present work, a distributed environment has been set up using RMI to estimate and to monitor load flow solutions for different sub-systems of an integrated power system. Each subsystem has been considered as a power system client and hence multi power system clients-single load flow server model is implemented. These power system clients are interconnected with a load flow server as shown in Fig. 1. When there is a call for a method located in a remote object, Fig. 1 shows the flow of remote method invocation.

A client computer basically does the distributed power system monitoring through an applet for every specific period of time and frequently exchanges data with the server. The server does the load flow computation and then distributes the results. Chronologically the server process should be started first, so that it can take the initiative to set up a connection link. It then waits until it receives a connection request from the client. A client can register itself with the remote object (server object), just by invoking the registration procedure on the server object, when it needs a service from it. The remote object obtains the necessary data from the registered client objects and responds back to them respectively with the results. The total process can be automated by making the server get the input data for every specific period of time. Transaction of data among clients and server takes place several times and so the possibilities of the occurrence of errors may be high. Hence it must be handled properly.

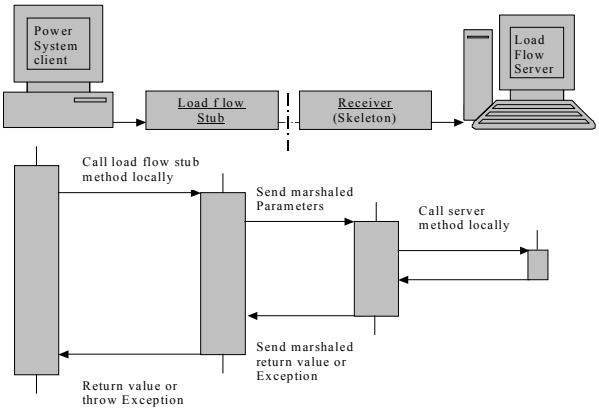

Fig 2. Invoking a load flow method on a remote object.

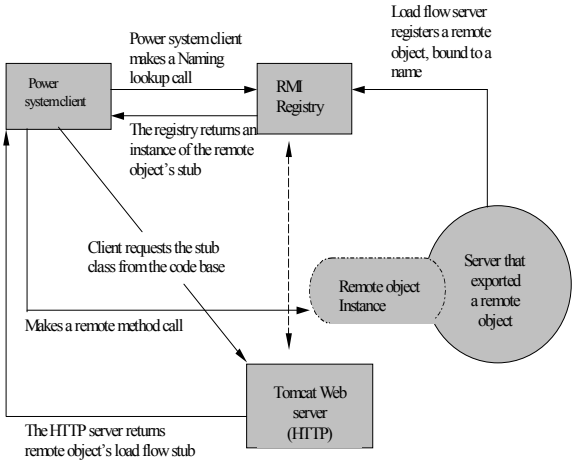

Fig. 3. Dynamic class loading.

A. RMI Data Flow Model

In the proposed model, each neighboring power system is considered as a client remote object. The power system client calls a method on an object that represents the remote object, which is called a stub. The stub contains a method for each of the methods in the remote object. Load flow stub always resides on the client's side and it packages the power system data into a block of bytes that can be communicated through the network. The process of marshalling [5] presents the entire load flow data in a suitable format for transporting one virtual machine to another. The load flow stub on the client's side builds the information block that consists of an identifier of the remote object to be used, a description of the method to be called and the marshalled load flow data. When the load flow stub sends the information to the load flow server, a receiver object (skeleton) on the server side receives and unmarshals the parameters. This receiver locates the object to be called and then calls the desired method with those parameters. When the method returns, the receiver object captures, marshals the return value and sends the marshalled load flow results as packets on to a marshal stream and thus sends the load flow result to the stub. The stub unmarshals the return value and returns it to the original caller. This data flow model is shown in Fig. 2.

B. Load Flow Server's Self Registry Service and Dynamic Class Loading

RMI provides bootstrap registry service to locate remote server objects. Server program registers remote objects with the bootstrap registry service and the clients retrieve stubs to those objects. In this proposed method, the load flow server creates its own registry and it maintains the stubs for the remote objects on its own and hence the server no longer needs to depend on the bootstrap registry service provided by RMI protocol.

In RMI client-server architecture, clients can communicate with the remote object only when the server side stub is available with the client. The stub can be loaded on the client's side dynamically by an external web server as shown in Fig. 3. The steps involved in downloading RMI stubs are as follows:

i. The remote object's codebase is specified by the remote object's server by setting the java.rmi.server.codebase property. The RMI load flow server registers a remote power system client object, bound to a name, with the RMI registry.
ii. The power system client makes a request for a reference to a named remote object. The reference to the remote object's stub instance is what the client will use to make remote method calls to the load flow server object.
iii. The RMI registry returns the stub instance reference to the requested class.
iv. The codebase which the power system client uses, is the URL that is annotated to the stub instance when the stub class was loaded by the registry.
v. The class definition for the stub is downloaded to the client dynamically.
vi. Now the power system client has all the information that it needs to invoke remote method on the load flow server object. The stub instance acts as a proxy to the remote object that exists on the server.

III. RMI Based Load Flow Monitoring Algorithm

Any changes in the implementation of the server side results in the modification of the stub and it will be made available for clients by dynamic class loading through an external server.

When a client's remote object registers with load flow server's remote object, the server uses the remote client reference to invoke its method to obtain the load flow data from that client and then provides the service through its methods. Both client and server objects are considered as remote objects and this is how inter-remote object communication is achieved. The server object uses a single thread of control to distribute the load flow solution simultaneously to the clients registered with it. The proposed model is dynamic which allows a new power system client to register with the load flow server object at run-time and to get serviced. A kind of multicasting has been achieved through this multi-client/single-server model. Load flow server and clients have to store, the necessary object codes required for load flow calculations. Stubs for both client and server must be kept at a common location like web server for distribution. Subsequently, the following steps are to be carried out:

i. Start load flow server.
ii. Load flow server should invoke its own registry service.
iii. Start power system client by dynamically loading the server's stub from the common location.

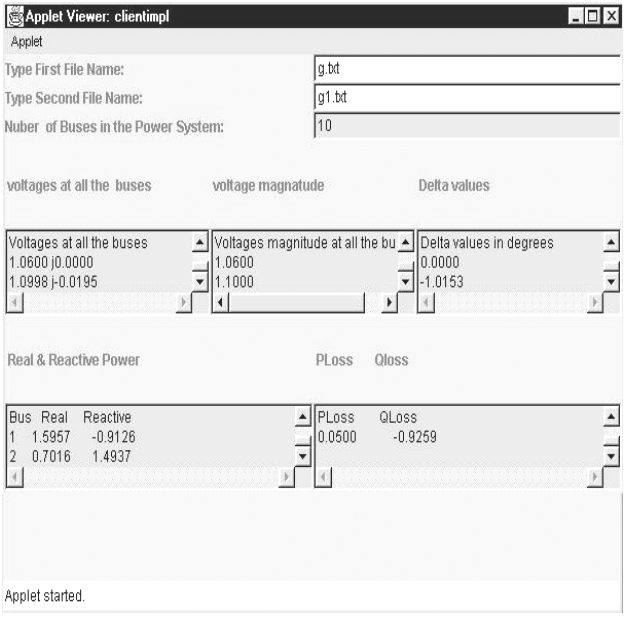

Fig. 4. Applet with load flow solution.

iv. Client registers with the server by invoking the appropriate method at the remote object.
v. Server uses the client's reference to receive the power system data from the client.
vi. Server computes the load flow result and returns it to the client.
vii. Client obtains the result from the server through load flow stub and provides a view of the result through an applet.
viii. For every specific period of time, server automatically receives system data from the client, thereby providing an automatic load flow monitoring.

IV. RESULTS

The above distributed algorithm has been implemented in Windows NT based HP workstations connected in an Ethernet LAN. The results are shown in a client applet as given in Fig. 4.

The above applet shows the load flow solution for a specific 10-bus power system client. When each power system client applet is loaded, it registers with the load flow server, the server stub will be downloaded dynamically and through it, the client sends the request and receives the output. Using this approach, different power system clients can monitor continuous updated load flow solutions at regular time intervals.

The major factors that influence the performance of on-line load flow monitoring in a distributed environment are Round Trip Time (RTT), scalability, reliability and security. The round trip time without overheads for the load flow analysis for different power system are measured, compared with the client/server technique and the result are shown in Fig. 5. RMI has the best performance in all the cases. It is clear that conventional client/server model is comparatively slower than RMI. The time taken by the clients increases steadily during the initial stages and increases slowly as the convergence time and the number of clients increases.

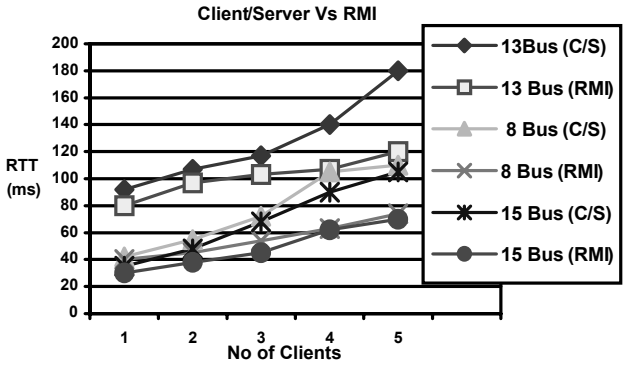

Fig. 5. RTT vs. no of clients.

V. CONCLUSION

An effective RMI based distributed model has been developed to monitor the load flow of multiple power systems. It has been tried out in overcoming the overheads associated with sequential power system load flow computation through this model. Although, client-server architecture for load flow solution is very well established, the value of this study lies in that it emphasizes a unique methodology based on remote method invocation to serve a large number of clients in a distributed power system environment, across various platforms based on communication between virtual machines. A practical implementation of this approach suggested in this paper was assessed based on 6, 8, 9, 10, 13 and 15 bus sample systems. Accordingly the proposed model can be implemented for large power systems network spread over geographically apart.

REFERENCES

[1] G. Bandyopandhyay, I. Senguptha, and T. N. Saha, "Use of client-server model in power system load flow computation," *IE(I) Journal-Electrical*, vol. 79, no. 1, pp 199-203, Feb. 1999.
[2] D. G. Hart, D. Uy, J. Northcote-Green, C. Laplace, and D. Novosel, "Automated solutions for distribution feeders," *IEEE* Trans. on *Computer Applications in Power*, vol. 15, no. 4, pp 25-30, Oct. 2000.
[3] B. Qiu and H. B. Gooi "Web based SCADA display systems (WSDS) for access via Internet," *IEEE Trans. on Power Systems*, vol. 15, no. 2, pp 681-686, May 2000.
[4] G. P. Azevedo, B. Feijo, and M. Costa "Control centers evolve with agent technology," *IEEE Trans. on Computer Applications in Power*, vol. 15, no. 3, pp 48-53, Jul. 2000.
[5] C. S. Horstmann and G. Cornell, *Core Java Volume II Advanced Features*, The Sun Microsystem Press Java Series, 2000.
[6] *Dynamic Code Downloading Using Remote Method Invocation*, http://java.sun.com/j2se/1.3/docs/gulde/rmi/codebase.html

K. Nithiyananthan received the B.E. degree in Electrical and Electronics Engineering and the M.E. degree in Power System Engineering from the Faculty of Engineering and Technology, Annamalai University, Chidambaram, India, in 1998 and 2000, respectively. He is currently working as a Teaching\Research Associate in the Department of Electrical and Electronics Engineering, College of Engineering, Guindy, Anna University, India. His research interests include power systems analysis and modeling, distributed computing and Internet technologies.

V. Ramachandran received his M.E. and Ph.D. in Electrical Engineering from College of Engineering, Guindy, Anna University, Chennai, India. He is currently a Professor of Computer Science and Engineering in College of Engineering, Guindy, Anna University, India. His research interests include power systems reliability engineering, network security, component technologies and soft computing.